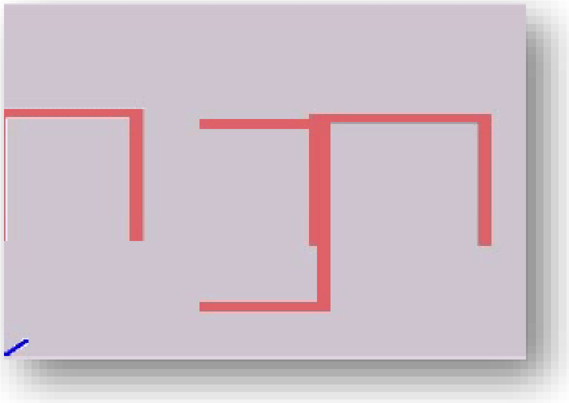
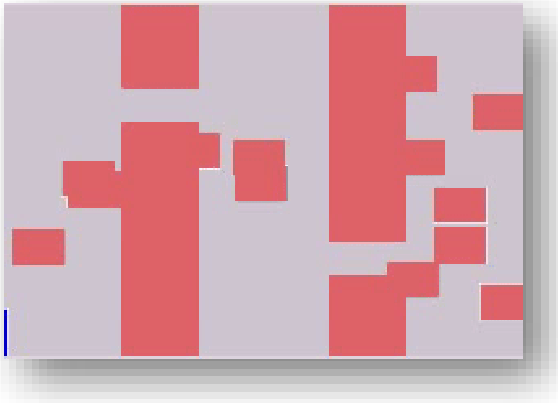
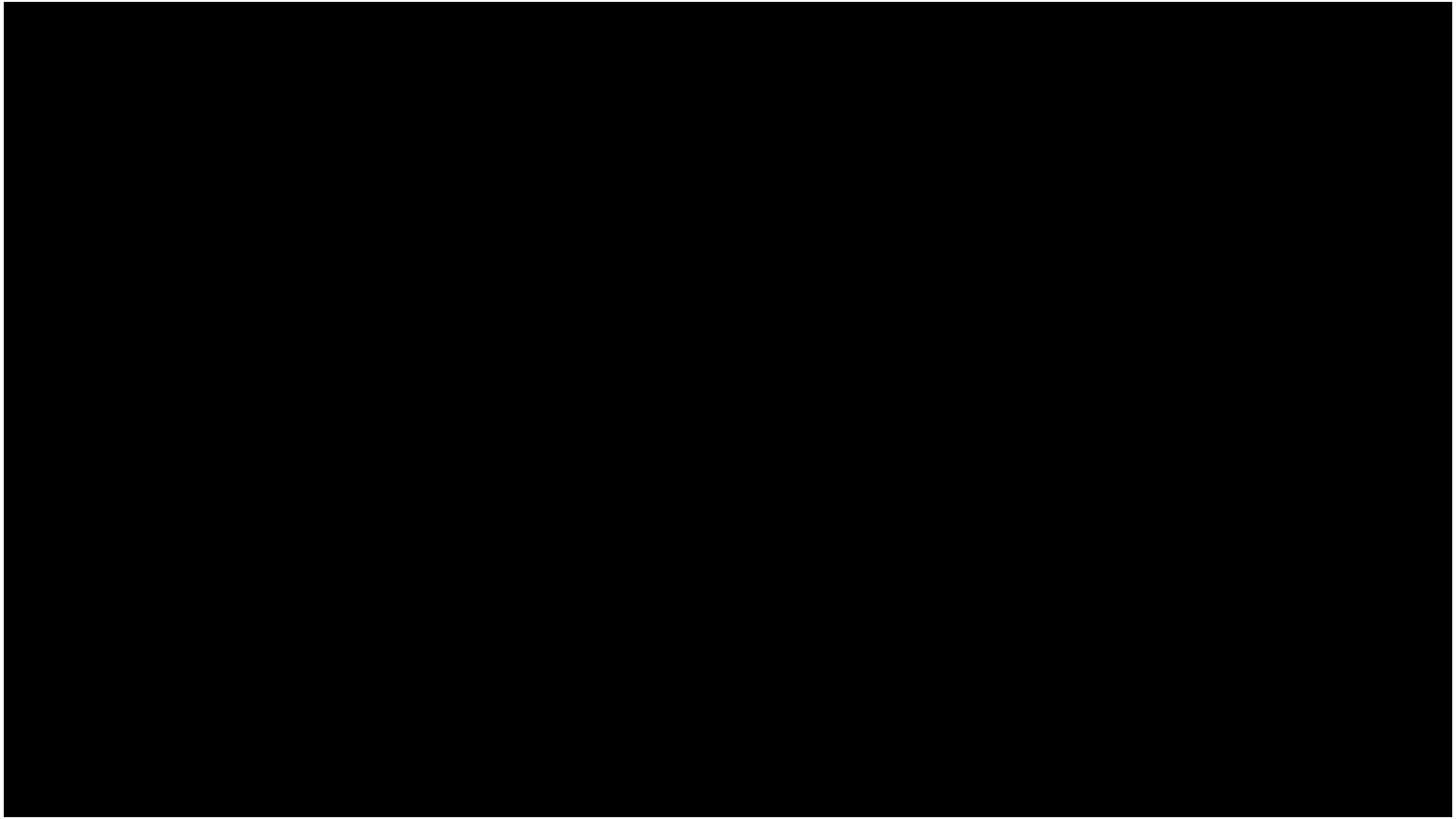
Learning Heuristic Search via Imitation

Mohak Bhardwaj, Sanjiban Choudhury, Sebastian Scherer



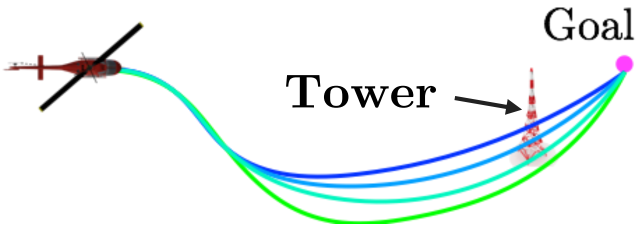


Real-time planning for fast UAVs

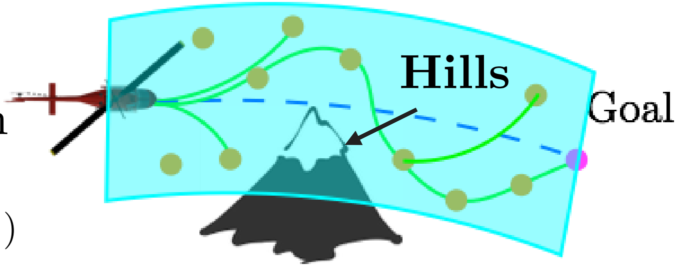


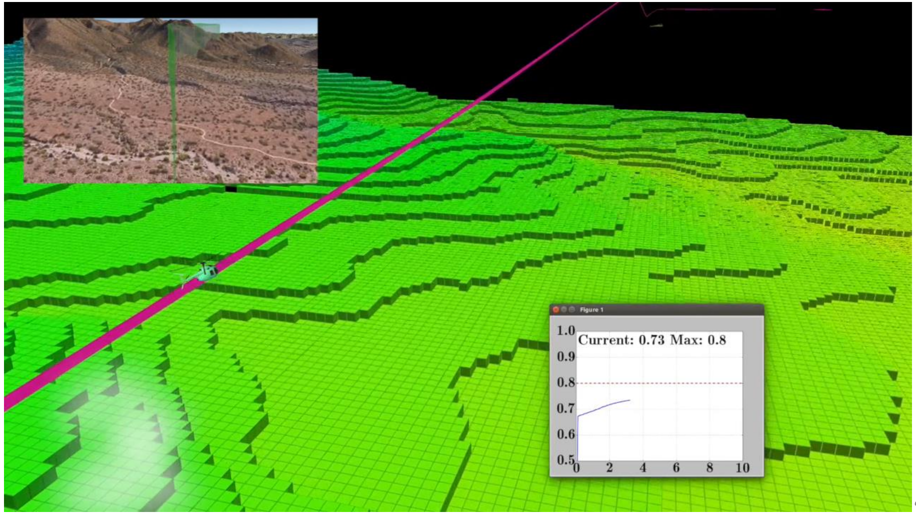
Different planners do well on different scenarios

Local trajectory optimization
(Ratliff et al.)



Informed sampling in RRT*
(Gammel et al.)

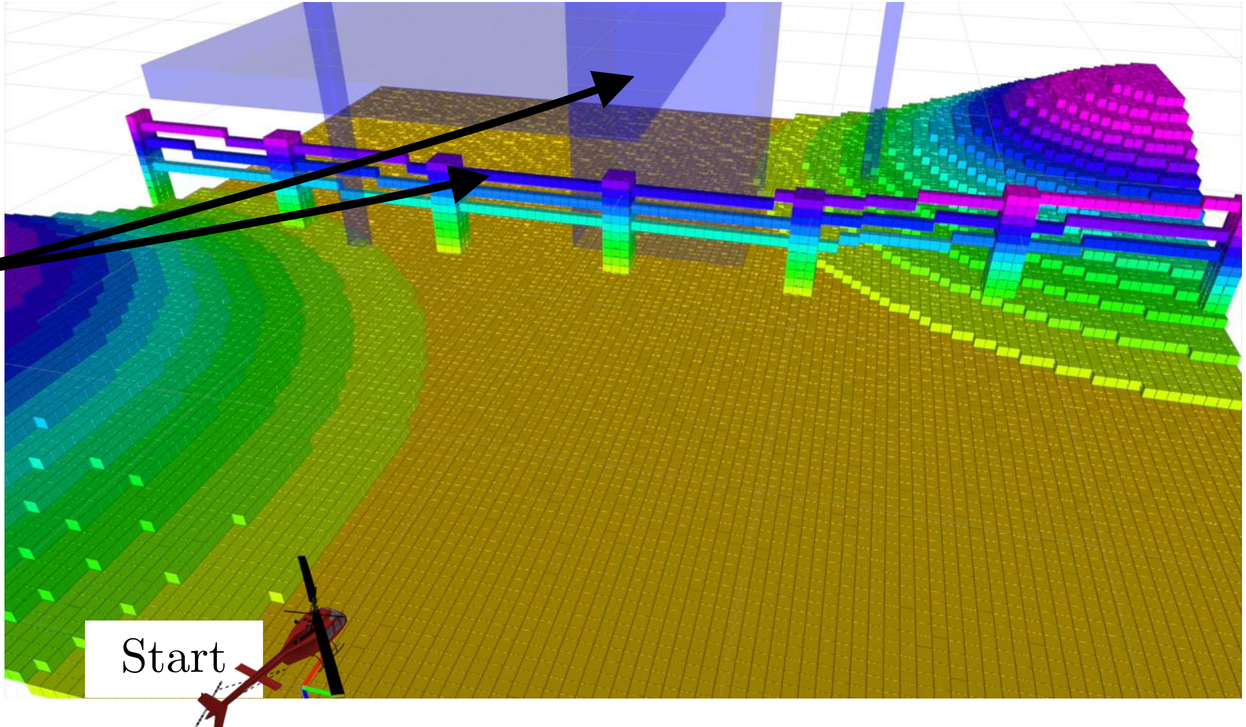




Can't we use a single versatile planner?

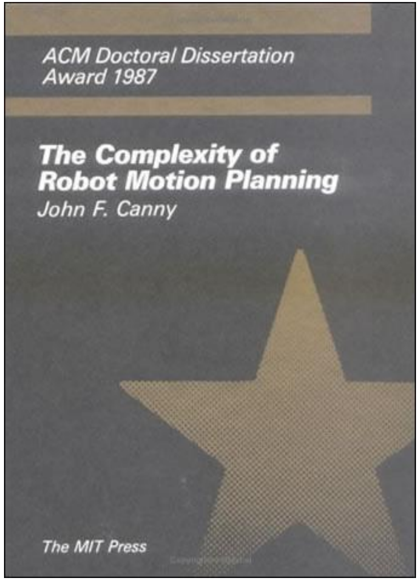
Optimal planners, by ignoring context, are unable to succeed in real-time

Wires and
no-fly-zones

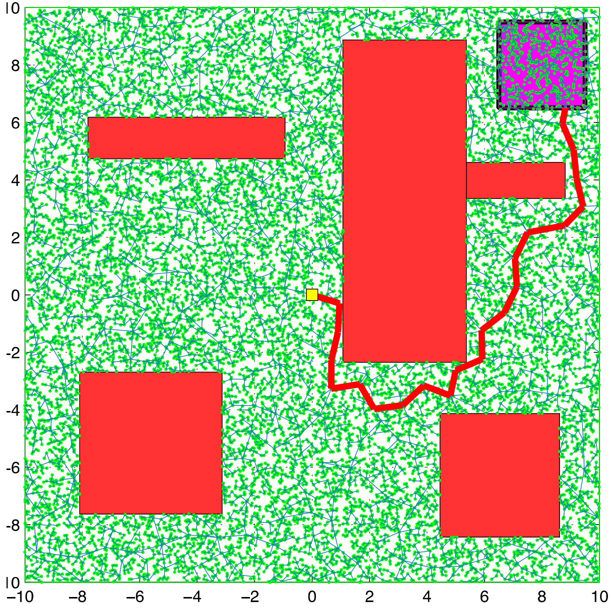


Planner
wastes time
checking
unlikely edges

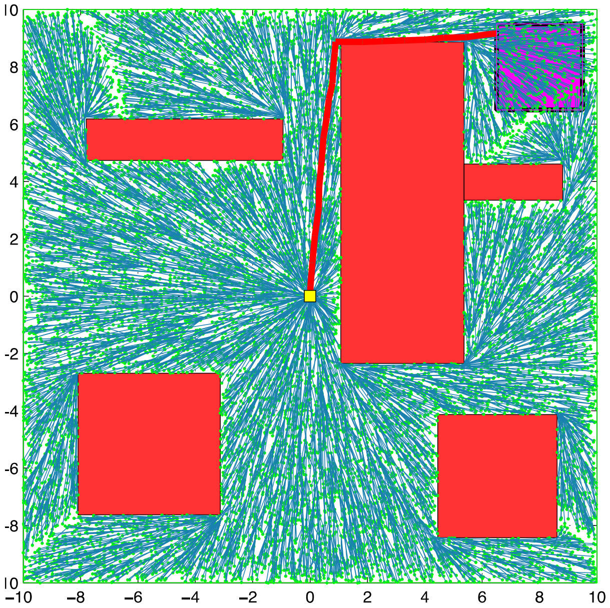
Historically, focus has been on worst-case



Computational complexity and completeness (Canny, 1988)



Probabilistic completeness (Kuffner and LaValle, 2000)

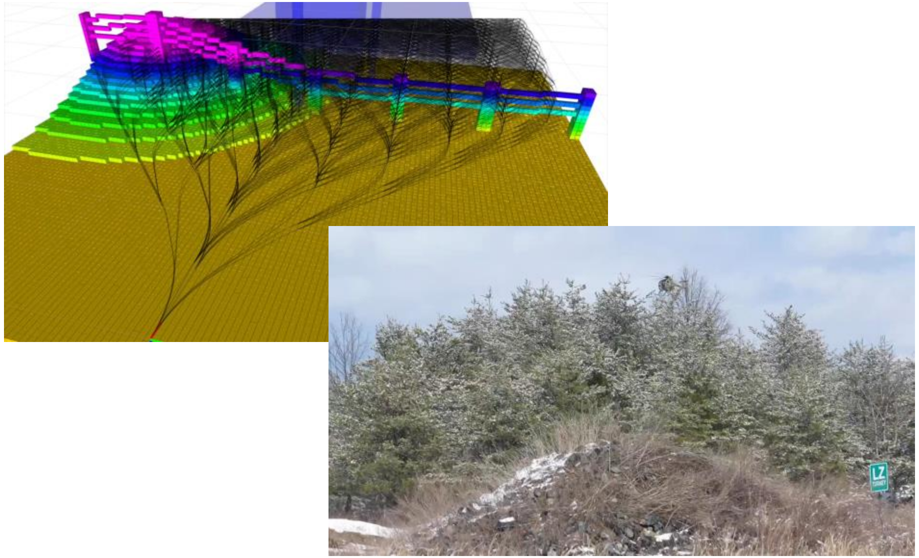


Asymptotic optimality (Karaman and Frazolli, 2010)

The case for data-driven planning

We should care about the expected performance of planners on the distribution of problems the robot actually encounters

Distribution of problems



→ **Train Planning Parameters** → Planner

We focus on the sub-problem of learning data-driven heuristics for graph search

Outline

1. Motivation: Why do we need heuristics in graph search?

2. Problem Formulation: Search as sequential decision making

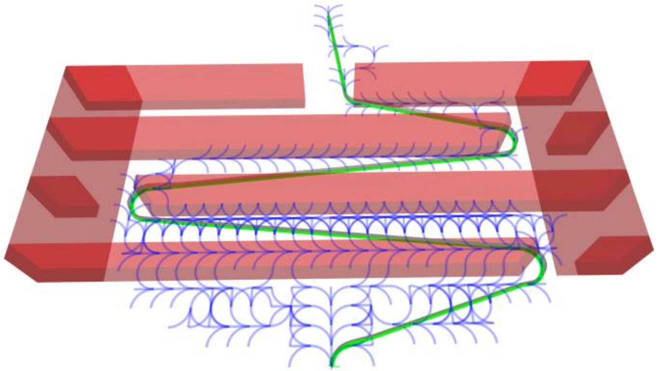
3. Approach: Training heuristic policies via imitation learning

4. Evaluation: Benchmark datasets, case studies, flight tests

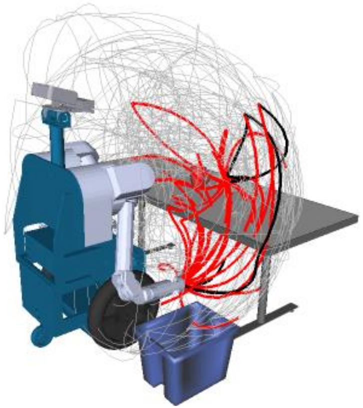
Outline

1. Motivation: Why do we need heuristics in graph search?
2. Problem Formulation: Search as sequential decision making
3. Approach: Training heuristic policies via imitation learning
4. Evaluation: Benchmark datasets, case studies, flight tests

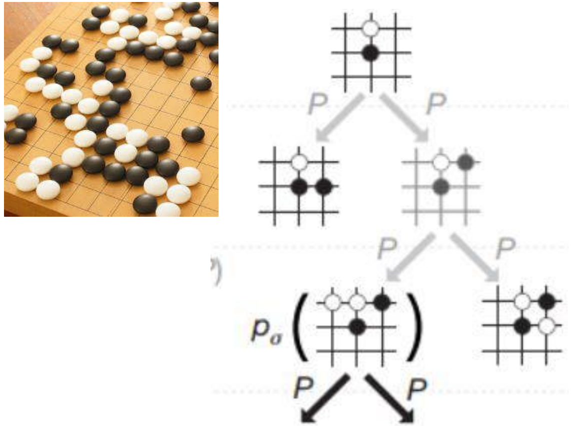
Graphs are excellent representations that allow generalization across domains



Non-holonomic path planning
(Our domain)



7D robot arm planning
(Dellin and Srinivasa, 2016)



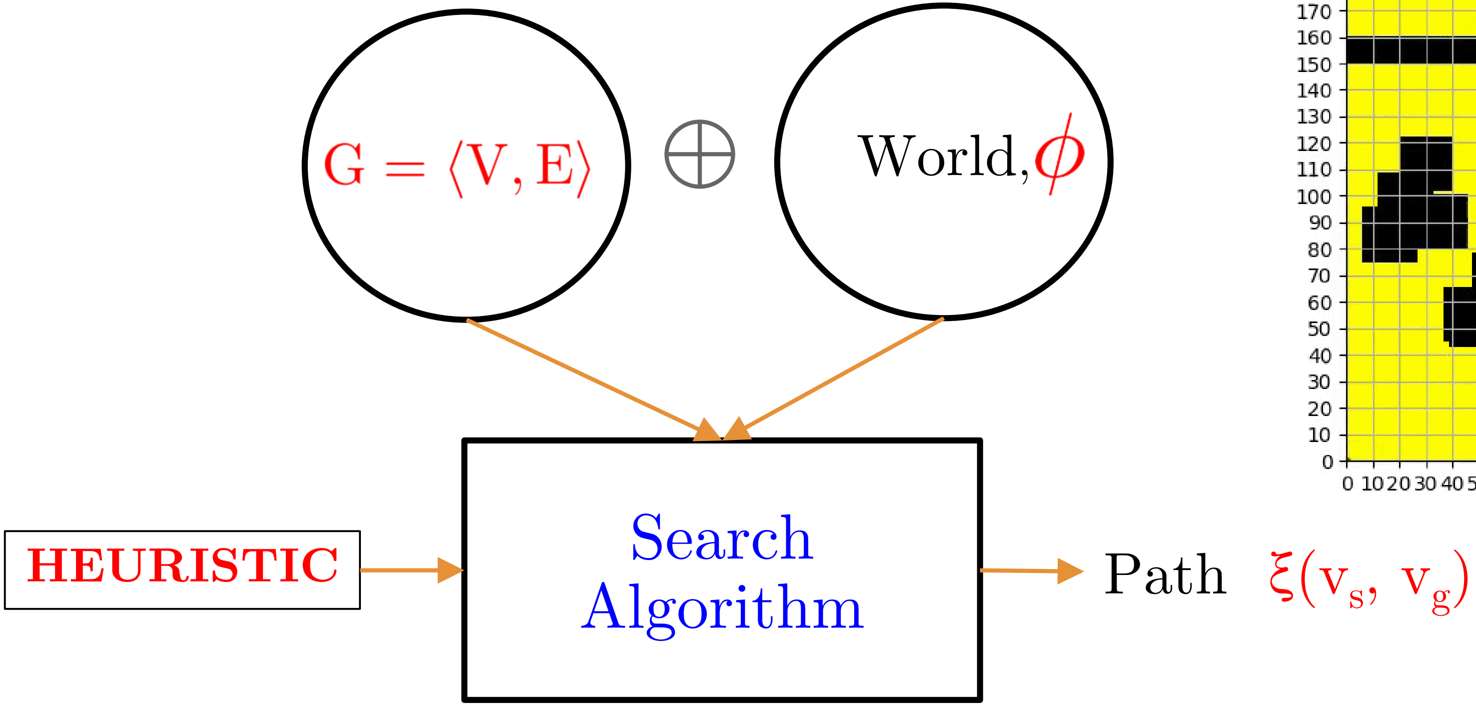
AlphaGo
(Silver et al., 2016)

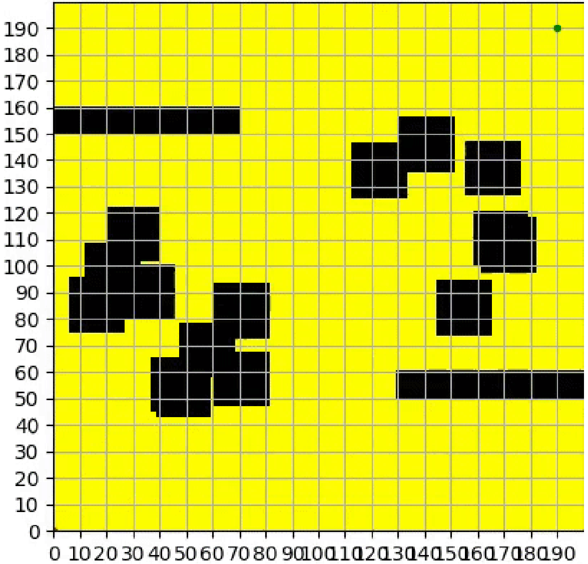
$$G = \langle V, E \rangle$$

Vertices: States of the robot

Edges: Dynamically feasible connections

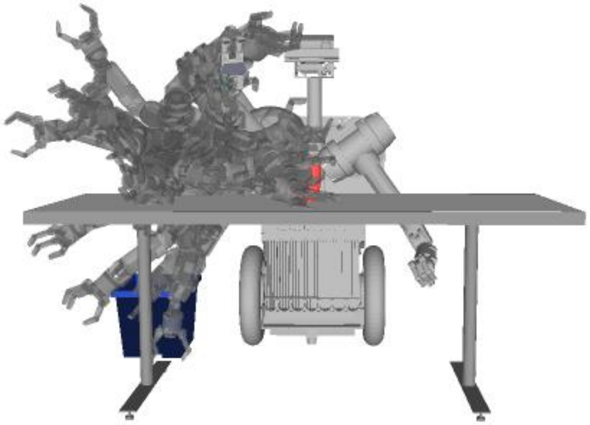
A heuristic guides the search tree



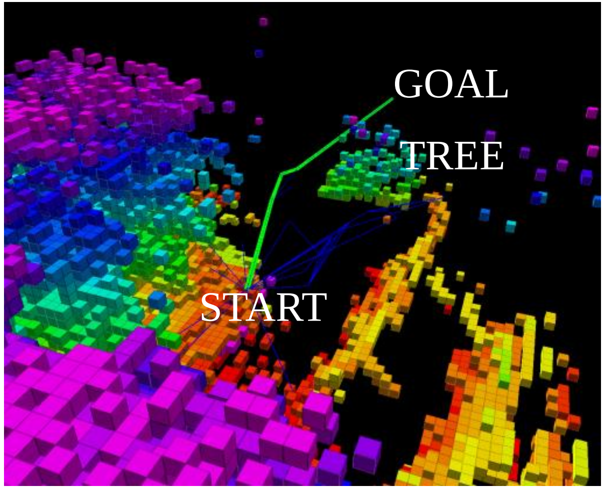


Heuristics should minimize edge evaluations

Online edge evaluation is the computational bottleneck in planning



Check robot mesh against all object meshes in environment



Check UAV volume against occupancy grid / point cloud

The key to real-time performance is minimizing online edge evaluations

Objective: Find a feasible path while minimizing edge evaluation

We want to compute a heuristic policy that explicitly minimizes expected edge evaluation

Finding a feasible path in real-time suffices for now

Can be extended to incorporate path cost in an anytime framework: Find a feasible path quickly and refine over time

Outline

1. Motivation: Why do we need heuristics in graph search?

2. **Problem Formulation:** Search as sequential decision making

3. Approach: Training heuristic policies via imitation learning

4. Evaluation: Benchmark datasets, case studies, flight tests

General framework for SEARCH

$\texttt{Search}\langle v_s, v_g, \texttt{Succ}, \texttt{Eval}, \phi, \texttt{Select}\rangle$

$\mathcal{O} \leftarrow v_s,\ \mathcal{C} \leftarrow \emptyset,\ \mathcal{I} \leftarrow \emptyset$

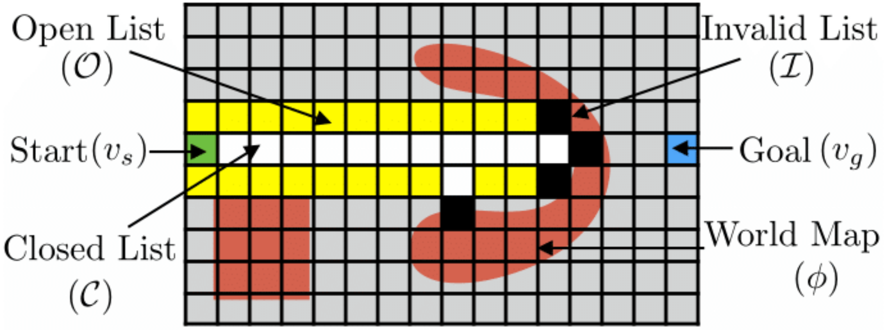
while $v_g \notin \mathcal{O}$:

$v \leftarrow \texttt{Select}\,(\mathcal{O})$

$(V_{\text{succ}}, E_{\text{inv}}) \leftarrow \texttt{Expand}\,(v, \texttt{Succ}, \texttt{Eval}, \phi)$

$\mathcal{O} \leftarrow \mathcal{O} \cup V_{\text{succ}},\ \mathcal{C} \leftarrow \mathcal{C} \cup v,\ \mathcal{I} \leftarrow \mathcal{I} \cup E_{\text{inv}}$

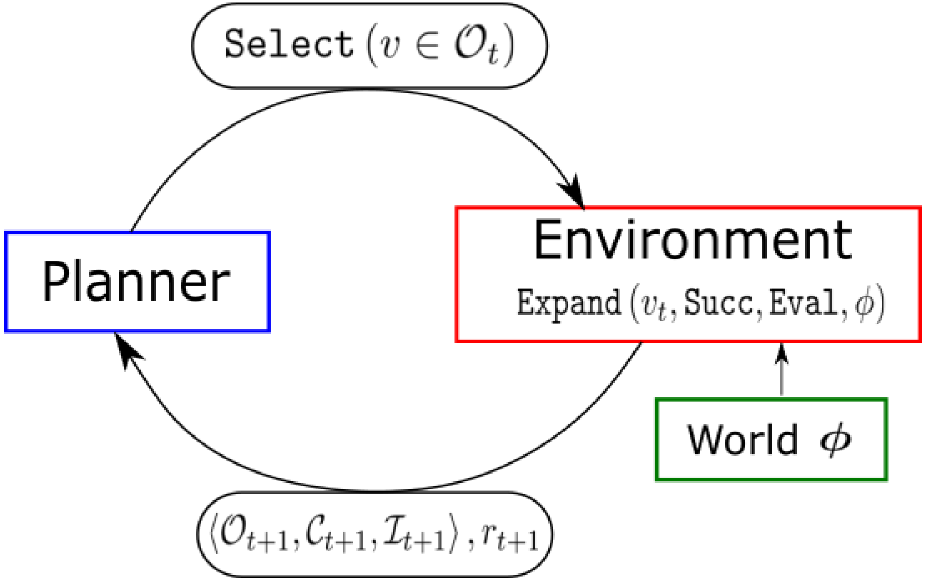
Return $\texttt{Path}\,(v_s, v_g)$



$\texttt{Select}\,(\mathcal{O})$ evaluates utility of each $v \in \mathcal{O}$ using heuristic function

ϕ is accessible to search only via collision checks i.e $\texttt{Eval}\,(\phi)$

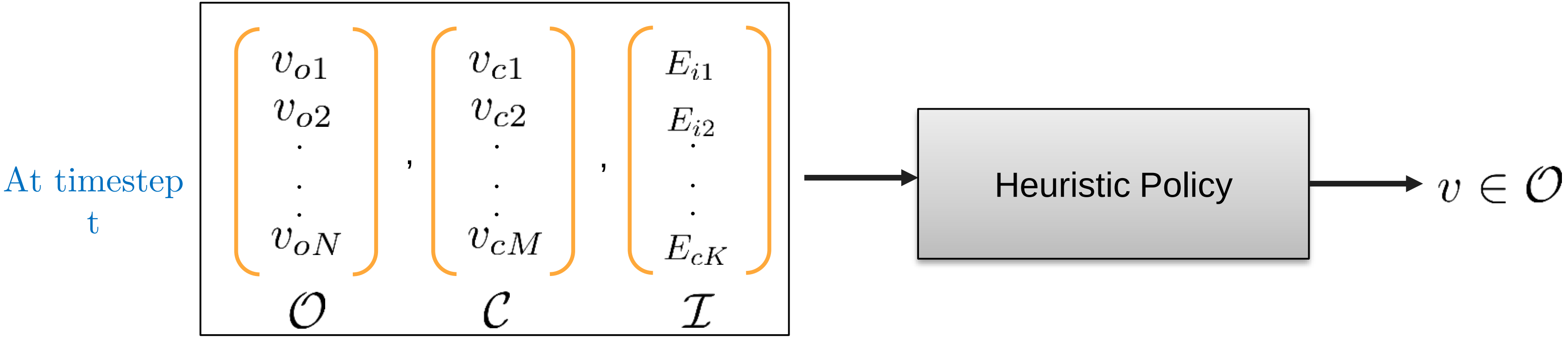
Key Insight: Search as sequential decision making under uncertainty(over World map)



State $\boldsymbol{s_t}$	$\langle \mathcal{O}_t, \mathcal{C}_t, \mathcal{I}_t \rangle$
Action $\boldsymbol{a_t}$	$\mathtt{Select}(v \in \mathcal{O}_t)$
Reward $\boldsymbol{r_t}$	0 if $v_g \in \mathcal{O}_t$ -1 otherwise
Transition Model $P(s_{t+1} \mid s_t, a_t)$	Induced by underlying world $\phi \sim P(\phi)$

Heuristics as policies

Classifier that maps state of search to node to expand (from Open).



Optimal policy explicitly minimizes planning effort.

Related Work

Learning heuristics for planning

Heuristics using supervised learning techniques

Yoon et. al, 2006

Xu et. al, 2007, 2009, 2010

Thayer et. al, 2011

Garrett et. al, 2016

Aine et. al, 2015

Deep Learning for planning

Incorporating long term deliberation in reinforcement learning and deep learning agents

Zhang et. al, 2016

Kahn et. al, 2014

Tamar et. al, 2016

Gupta et. al, 2017

Gao et. al, 2017

Imitation Learning of oracles

Non i.i.d supervised learning from oracle demonstrations under own state distribution

Ross et. al, 2011, 2014

Chang et. al, 2015

Sun et. al, 2017

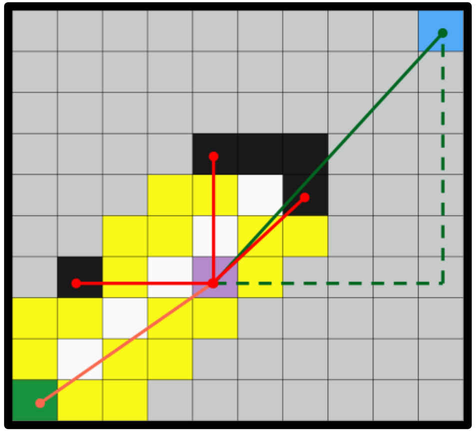
Choudhury et. al, 2017

Outline

1. Motivation: Why do we need heuristics in graph search?
2. Problem Formulation: Search as sequential decision making
3. **Approach:** Training heuristic policies via imitation learning
4. Evaluation: Benchmark datasets, case studies, flight tests

Representing search state

Compress search state $s_t = \langle \mathcal{O}_t, \mathcal{C}_t, \mathcal{I}_t \rangle$ to get $\boldsymbol{f_t}$ for each $v \in \mathcal{O}_t$



Search based: Depend on the current status of the search tree

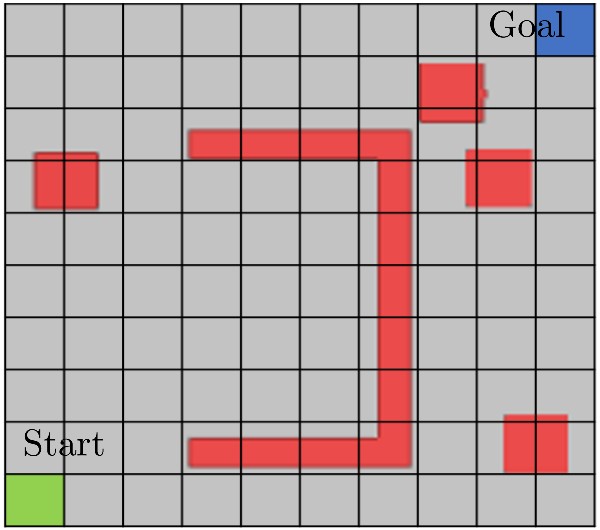
(x_v, y_v)	Vertex location in world	$\mathtt{h}_{\mathtt{EUC}}$	Euclidean distance to goal
(x_{v_g}, y_{v_g})	Vertex location in world	$\mathtt{h}_{\mathtt{MAN}}$	Manhattan distance to goal
g_v	Cost of shortest path to start	d_{TREE}	Vertex depth in tree

World based: Depend on environment uncovered so far

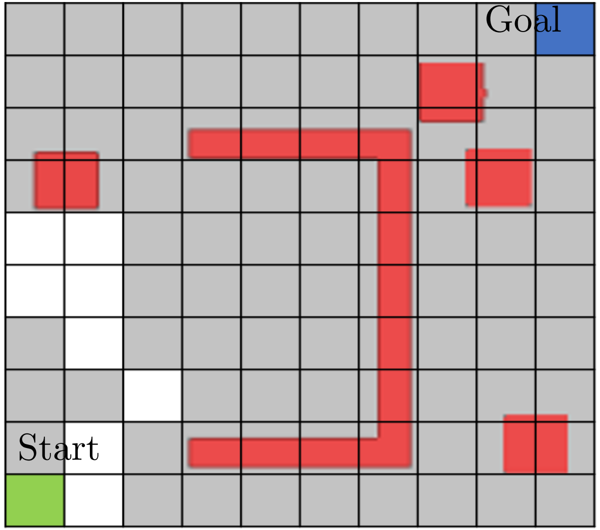
$(x_{\mathrm{OBS}}, y_{\mathrm{OBS}}, d_{\mathrm{OBS}})$	Coordinates and location of closest node in $\mathcal{I}$
$(x_{\mathrm{OBSX}}, y_{\mathrm{OBSX}}, d_{\mathrm{OBSX}})$	Coordinates and location of closest node in $\mathcal{I}$ in x-coordinate
$(x_{\mathrm{OBSY}}, y_{\mathrm{OBSY}}, d_{\mathrm{OBSY}})$	Coordinates and location of closest node in $\mathcal{I}$ in y-coordinate

Note: Feature calculation should not expend extra search effort!

Model-free reinforcement learning is slow to converge



Input problem



Poor rollout with learner

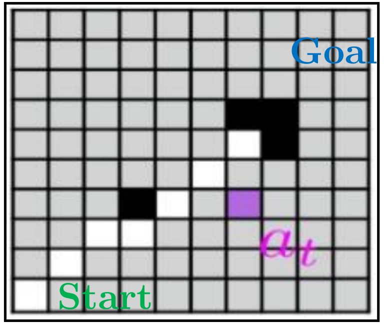
White – Nodes Expanded

Large state and action spaces; Sparse reward

But we can do better!

Key Insight: Construct an optimal oracle using dynamic programming. (*backward Dijkstra's algorithm*)

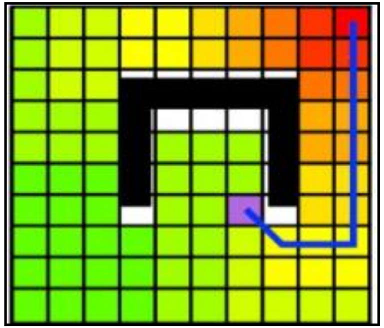
Learner



Approximates search effort from belief

IMITATION

Oracle



Solves full problem to get *true* expansions-to-go

Oracle is "**clairvoyant**" with access to true state of underlying world. (Choudhury et al., 2017)

Imitation Learning with cost-to-go

Learn a function approximator for the oracle's Q value

$$\hat{\theta} = \underset{\theta \in \Theta}{\arg\min} \; \mathbb{E}_{\substack{\phi \sim P(\phi) \\ \boxed{t \sim \mathcal{U}(1 \ldots T)} \\ \boxed{s \sim d_\pi^t}}} \left[\left(Q_\theta\left(s_t, a_t\right) - \boxed{Q^{\mathrm{OR}}\left(v, \phi\right)} \right)^2 \right]$$

Uniformly sampled time-step

Distribution of states under roll-in policy π

Oracle label

Planner follows greedy policy with respect to search effort

$$\hat{\pi}(s_t) = \underset{a_t \in \mathcal{A}}{\arg\min} \; Q_{\hat{\theta}}\left(s_t, a_t\right)$$

Reduction to no-regret online learning

Learn a function approximator for the oracle's Q value

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \mathbb{E}_{\substack{\phi \sim P(\phi) \\ \boxed{t \sim \mathcal{U}(1 \ldots T)} \\ \boxed{s \sim d_\pi^t}}} \left[\left(Q_\theta(s_t, a_t) - \boxed{Q^{\mathrm{OR}}(v, \phi)} \right)^2 \right]$$

Uniformly sampled time-step

Distribution of states under roll-in policy π

Oracle label

Problem:

Using oracle to roll-in leads to distribution mismatch

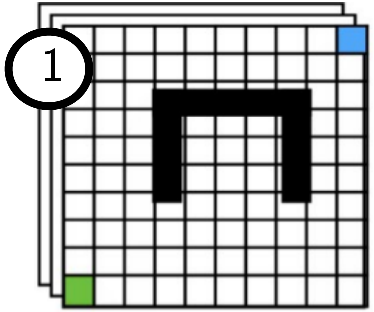
Solution:

Iterative learning, roll-in with mixture of oracle + learner, dataset aggregation

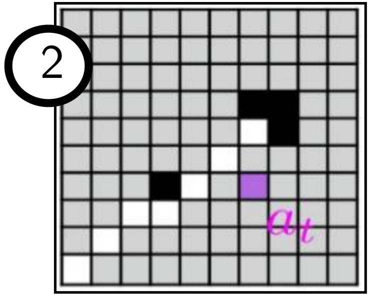
(Ross and Bagnell, 2014)

Search As Imitation Learning (SAIL)

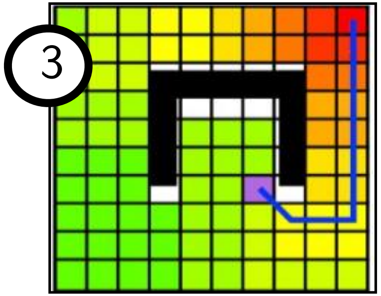
Run m episodes in every iteration $i = 1 \ldots N$



Sample problem



Roll-in mixture; choose random action; collect $\langle \mathcal{O}_t, \mathcal{C}_t, \mathcal{I}_t, \boldsymbol{v_t} \rangle$



Query oracle for $\boldsymbol{Q^{OR}(v, \phi)}$

$$\mathcal{D} \leftarrow \mathcal{D} \cup \left\langle v_t, s_t, Q^{\mathrm{OR}} \right\rangle$$

Repeat steps (2-3) at k uniformly sampled steps

Train π_{i+1} on aggregated dataset $\mathcal{D}$

Repeat above steps to train N policies $\pi_1 \ldots \pi_N$

Return best π_i on validation

Outline

1. Motivation: Why do we need heuristics in graph search?
2. Problem Formulation: Search as sequential decision making
3. Approach: Training heuristic policies via imitation learning
4. **Evaluation:** Benchmark datasets, case studies, flight tests

Benchmark experiments: Setup

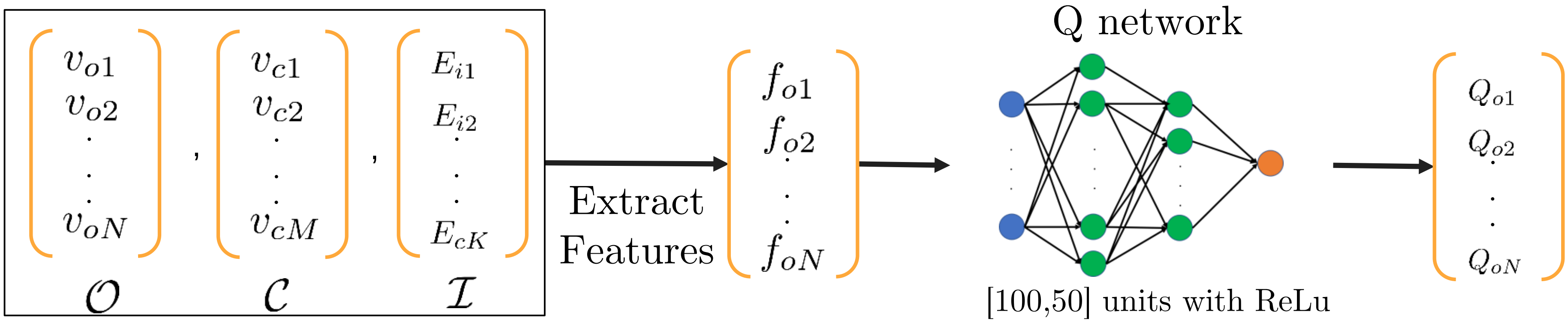
Code

8 different databases of 2D planning problems of varying complexity.

World: bitmap of obstacles and free space. Size: 200mx200m

Start and goal fixed across problems(bottom-left to top-right).

Graph, $G = \langle V, E \rangle$: 1m resolution and 8-connected neighbors.



Code and details: https://mohakbhardwaj.github.io/SaIL/

Benchmark experiments: Baselines

Motion Planning Baselines:

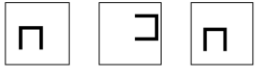
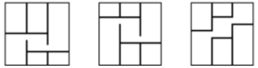
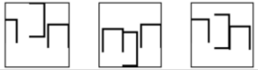
1. Greedy search with Euclidean heuristic (h_{EUC})
2. Greedy search with Manhattan heuristic (h_{MAN})
3. MHA* $([h_{EUC}, h_{MAN}, d_{OBS}])$

Machine Learning Baselines:

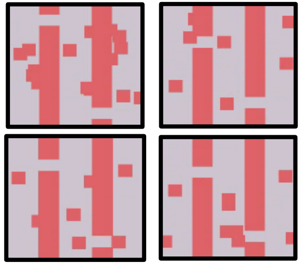
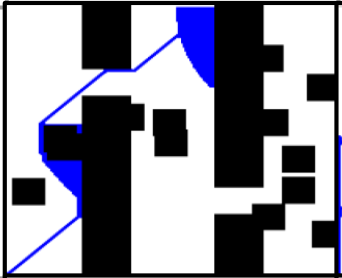
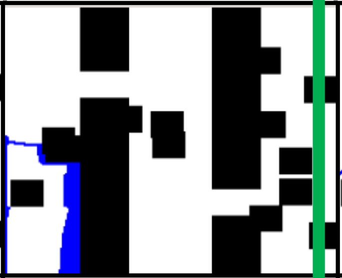
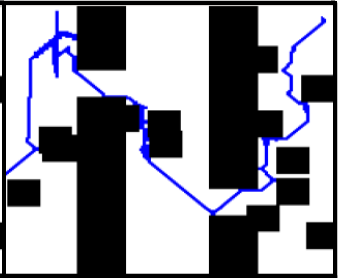
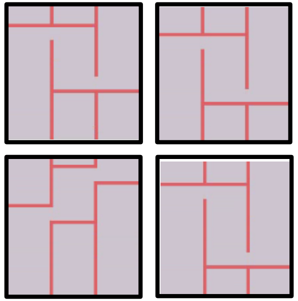
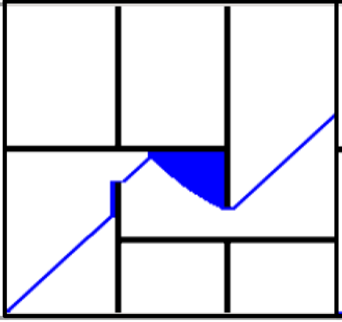
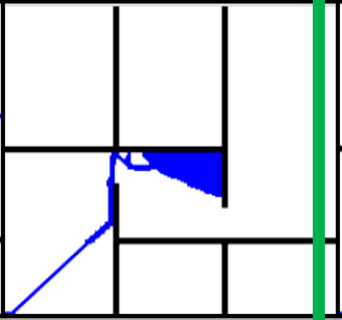
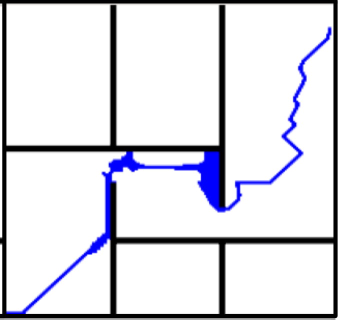
1. Behavior Cloning
2. Reinforcement Learning – C.E.M and Q-Learning.

All results shown are after 15 iterations of SAIL, training on 200 environments per iteration. Behavior Cloning trains on 600 environments

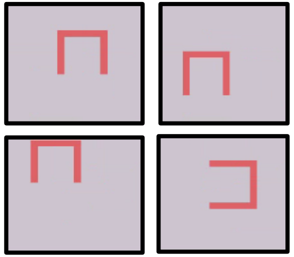
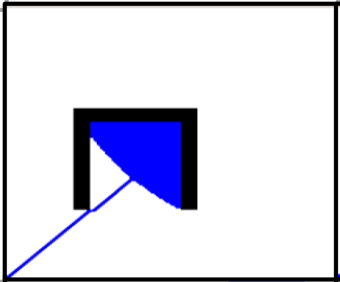
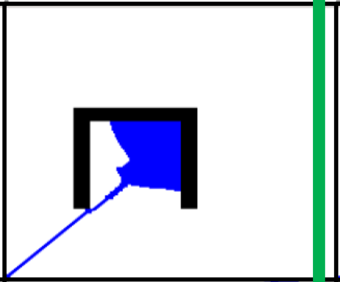
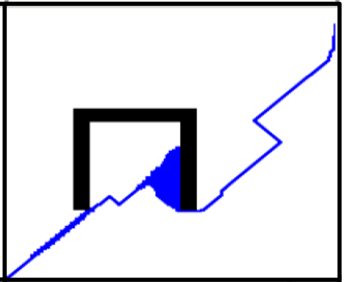
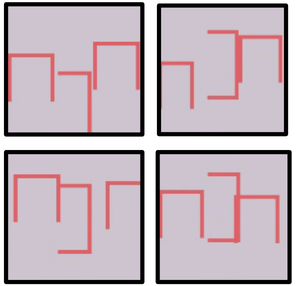
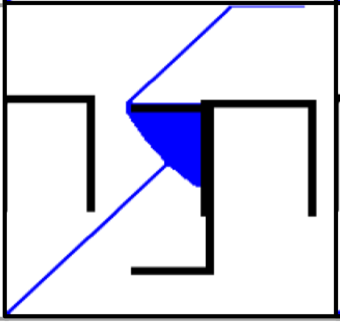
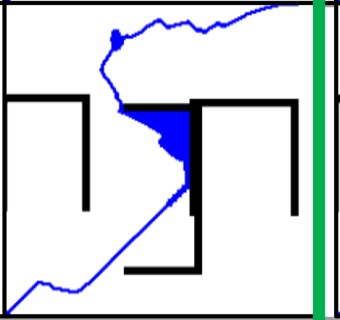
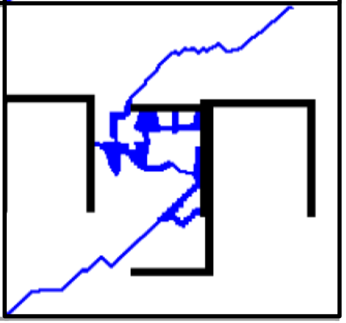
SaIL has competitive performance across all datasets

Dataset	Sample Worlds	SaIL	SL	CEM	QL	h_{EUC}	h_{MAN}	A*	MHA*
Alternating Gaps		**0.039**	0.432	0.042	1.000	1.000	1.000	1.000	1.000
Single Bugtrap		0.158	0.214	**0.057**	1.000	0.184	0.192	1.000	0.286
Shifting Gaps		**0.104**	0.464	1.000	1.000	0.506	0.589	1.000	0.804
Forest		**0.036**	0.043	0.048	0.121	0.041	0.043	1.000	0.075
Bugtrap+Forest		**0.147**	0.384	0.182	1.000	0.410	0.337	1.000	0.467
Gaps+Forest		**0.221**	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Mazes		**0.103**	0.238	0.479	0.399	0.185	0.171	1.000	0.279
Multiple Bugtraps		**0.479**	0.480	1.000	0.835	0.648	0.617	1.000	0.876

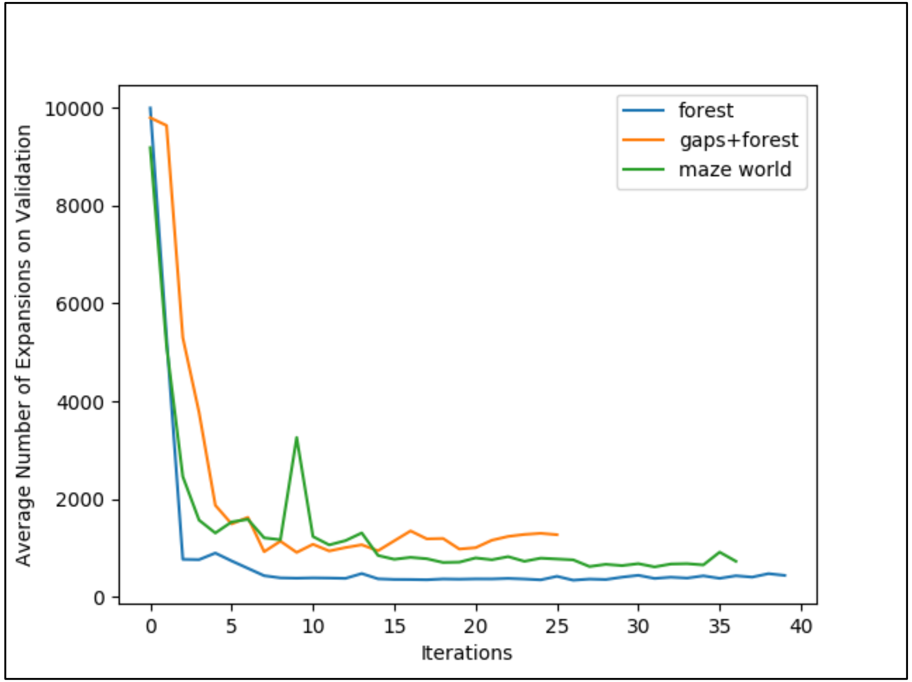
SaIL is able to exploit relative configuration of obstacles and environment structure.

	Dataset	Greedy (h_{EUC})	Behavior Cloning	SaIL
Gaps+Forest				
Mazes				

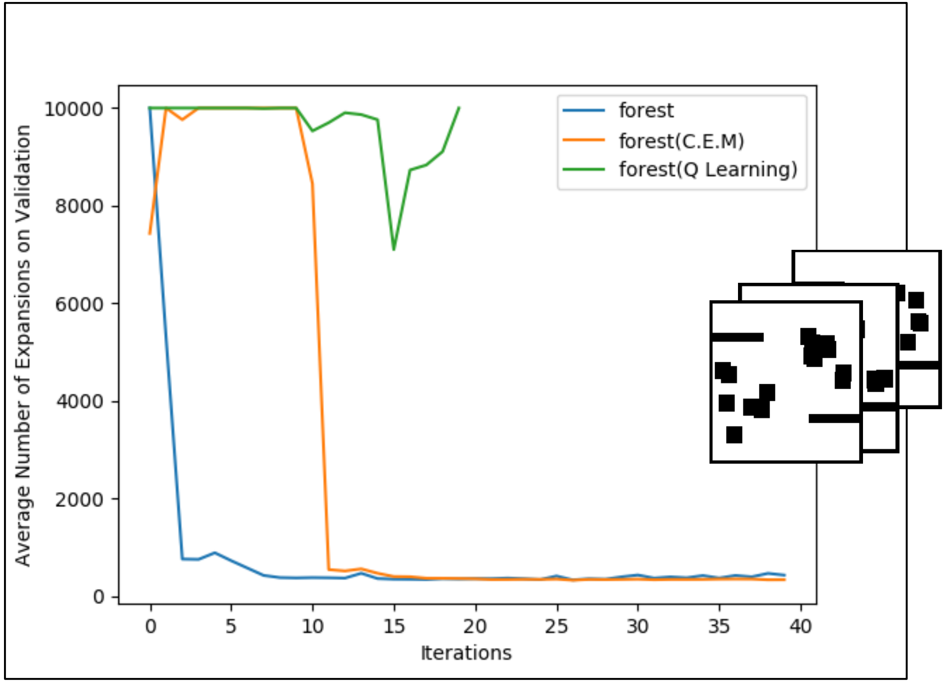
SaIL is able to detect and escape local minima

	Dataset	Greedy (h_{EUC})	Behavior Cloning	SaIL
Single Bug-trap				
Multiple Bug-traps				

SaIL has faster convergence than all learning baselines



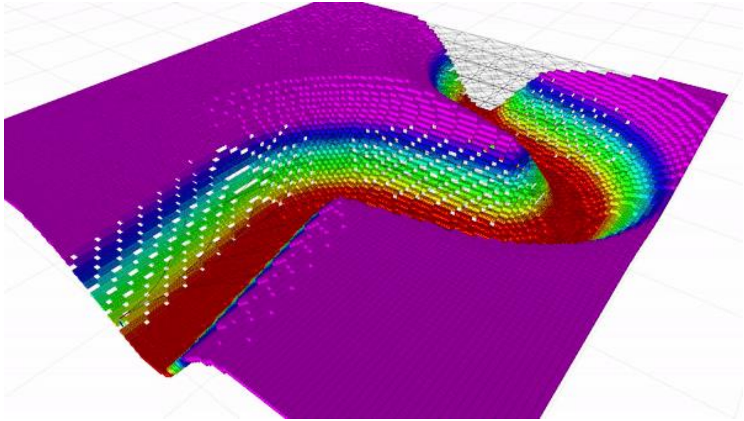
Converges fast consistently across environments

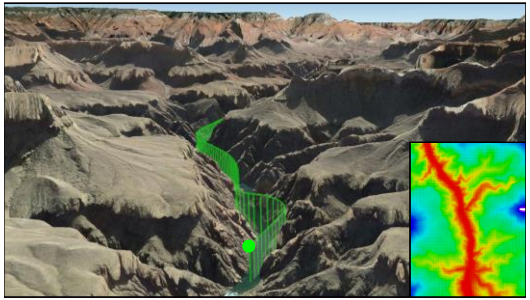


Converges way faster than model free RL

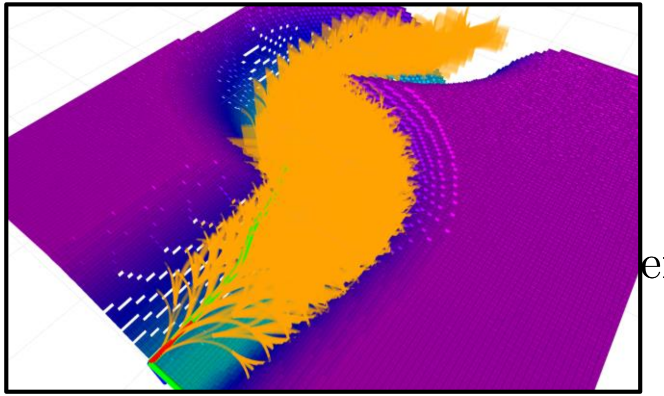
Current Work: Evaluation on helicopter planning

Dataset of canyons
(in simulation)





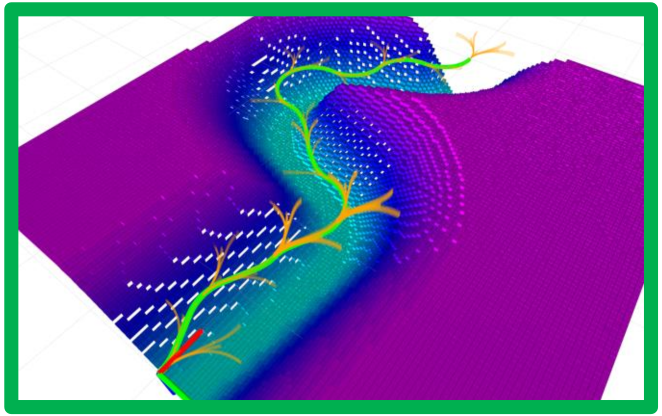
A*
(w=1)



Fills up
entire canyon

2532 expansions, 700ms

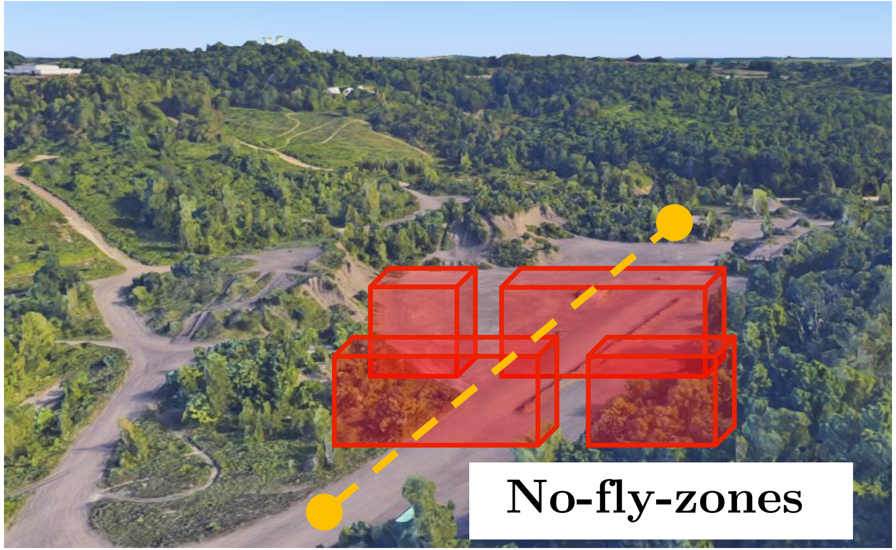
SAIL



Sticks to
middle of
canyon

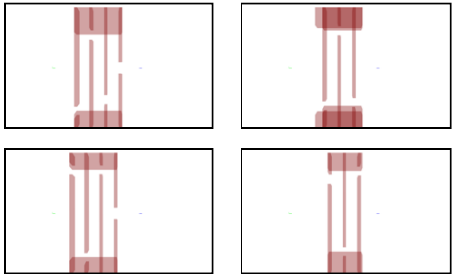
18 expansions, 100ms

Current Work: Evaluation on an UAV flying in complex environments

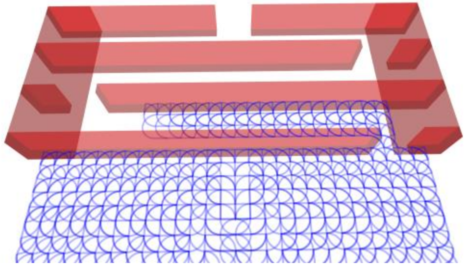


UAV has to fly at high speeds (5 - 15 m/s) and avoid no-fly-zones (other aircrafts / above building) that create complex environments

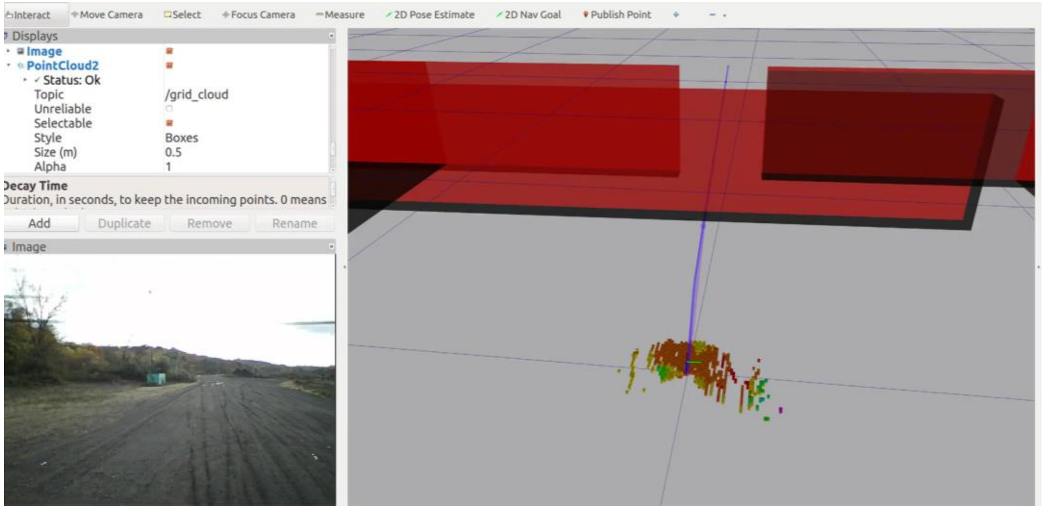
Evaluation on an UAV flying in complex environments

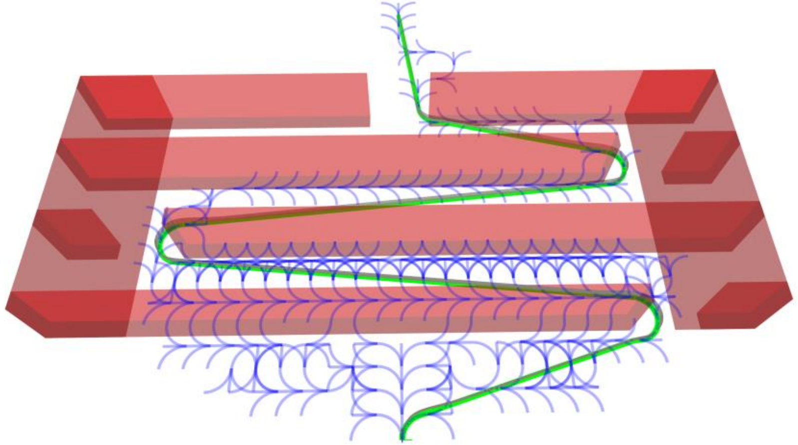


(Left) SaIL trained on dataset of mazes in simulation. (Below) Tested on a real maze with planning onboard



(Left) A* expands 1910 states (1000 ms). (Below) SaIL expands 180 states (120 ms)

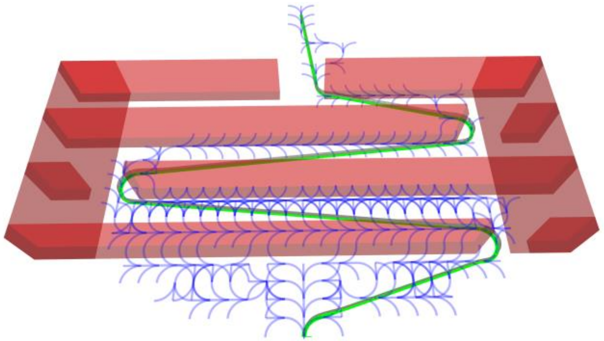


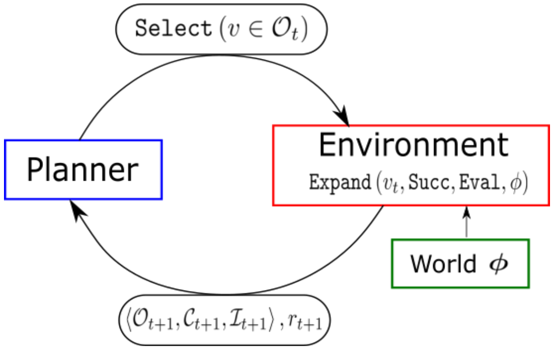


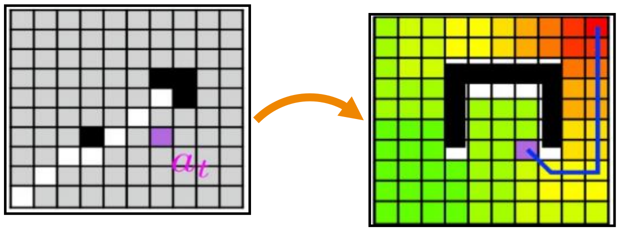
Summary

Code

Key Takeaways







IMITATION

Future Work

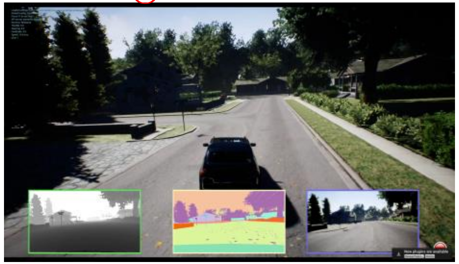
Recurrent architectures

Exploit the temporal structure of the problem and reduce dependence on features. (Deeply AggreVaTeD, Sun et. al, 2017)

Anytime Planning

Try to incorporate solution cost into heuristic training procedure.(Densification Strategies for Anytime Motion Planning over Large Dense Roadmaps, Choudhury et al, 2017)

Generating data for training



Microsoft AirSim

Appendix 1: Cost-Sensitive Imitation Learning

Learner's misclassification weighted by Oracle's Q-value (Ross et al., 2014):

$$\hat{\pi}(s) = \arg\min_{\pi \in \Pi} \mathbb{E}_{\substack{\phi \sim P(\phi) \\ t \sim \mathcal{U}(1 \ldots T) \\ s \sim d_\pi^t}} \left[Q^{\mathrm{COR}}(\pi(s), \phi) - \min_{v \in \mathcal{O}} Q^{\mathrm{COR}}(v, \phi) \right]$$

Cost-sensitive classification loss

$\mathrm{Q}^{COR}(v, \phi)$ - Oracle label for optimal number of expansions left

$\pi_{COR}(s_t, \phi) = \arg\min_{v \in \mathcal{O}} \left[\mathrm{Q}^{COR}(v, \phi) \right]$ - Optimal oracle policy

d_π^t - Distribution of states induced by rolling-in with mixture policy π

Use reduction of c.s classification to regression

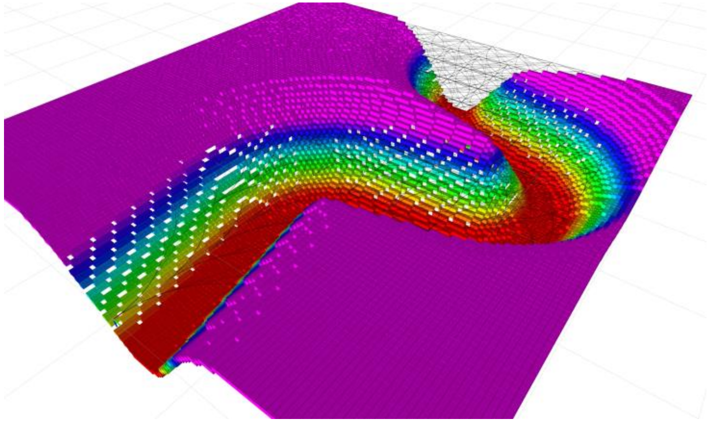
$$\hat{\theta} = \arg\min_{\theta \in \Theta} \mathbb{E}_{\substack{\phi \sim P(\phi) \\ t \sim \mathcal{U}(1 \ldots T) \\ s \sim d_\pi^t}} \left[\left(Q_\theta \left(s_t, a_t \right) \; - \; Q^{\mathrm{COR}} \left(v, \phi \right) \right)^2 \right]$$

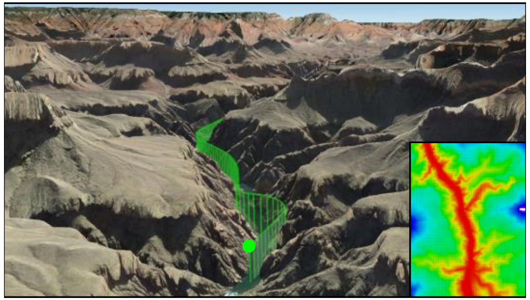
Planner greedily chooses node with least expected search effort

$$\hat{\pi}(s_t) = \arg\min_{a_t \in \mathcal{A}} Q_{\hat{\theta}} \left(s_t, a_t \right)$$

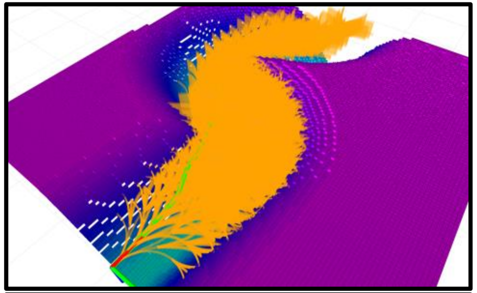
Appendix 2: Complete results of helicopter evaluation

Dataset of canyons



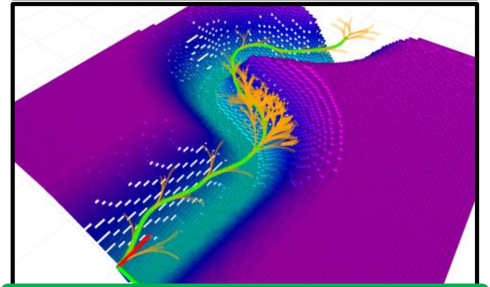


A*
(w=1)



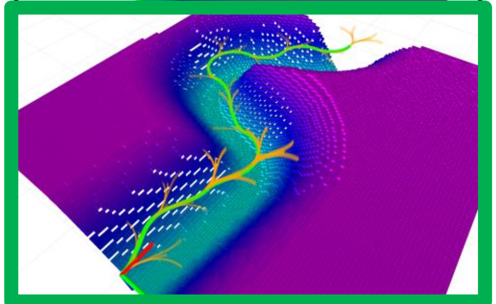
Fills up entire canyon

A*
(w=3)



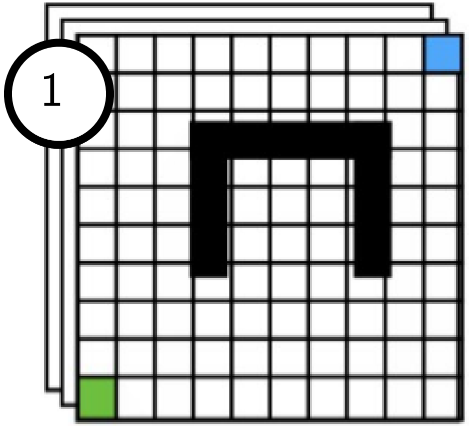
Local Minima at sharp turn

SAIL

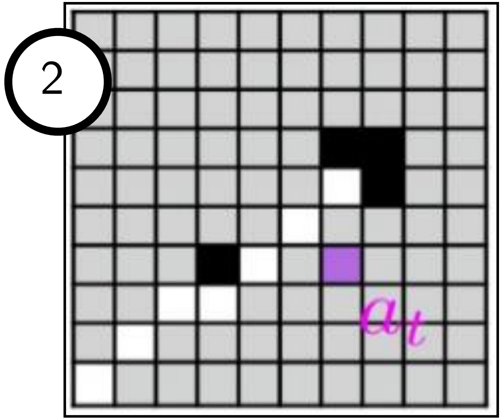


Sticks to middle of canyon

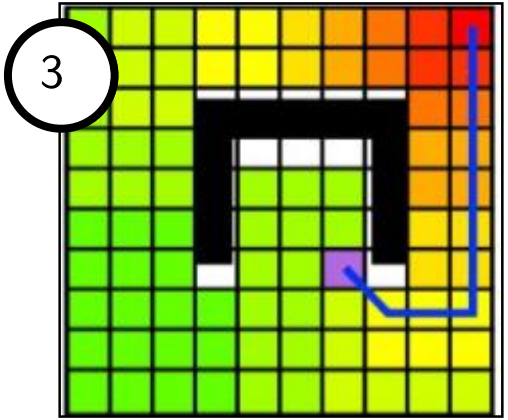
Appendix 3: SaIL algorithm steps



Sample a world



Roll-in mixture policy;
choose random action;
collect $\langle \mathcal{O}, \mathcal{C}, \mathcal{I}, \boldsymbol{v} \rangle$



Query oracle planner
for $\boldsymbol{Q^{OR}}(\boldsymbol{v}, \boldsymbol{\phi})$

4

$\mathcal{D} \leftarrow \mathcal{D} \cup \langle v_t, s_t, Q^{\mathrm{OR}} \rangle$

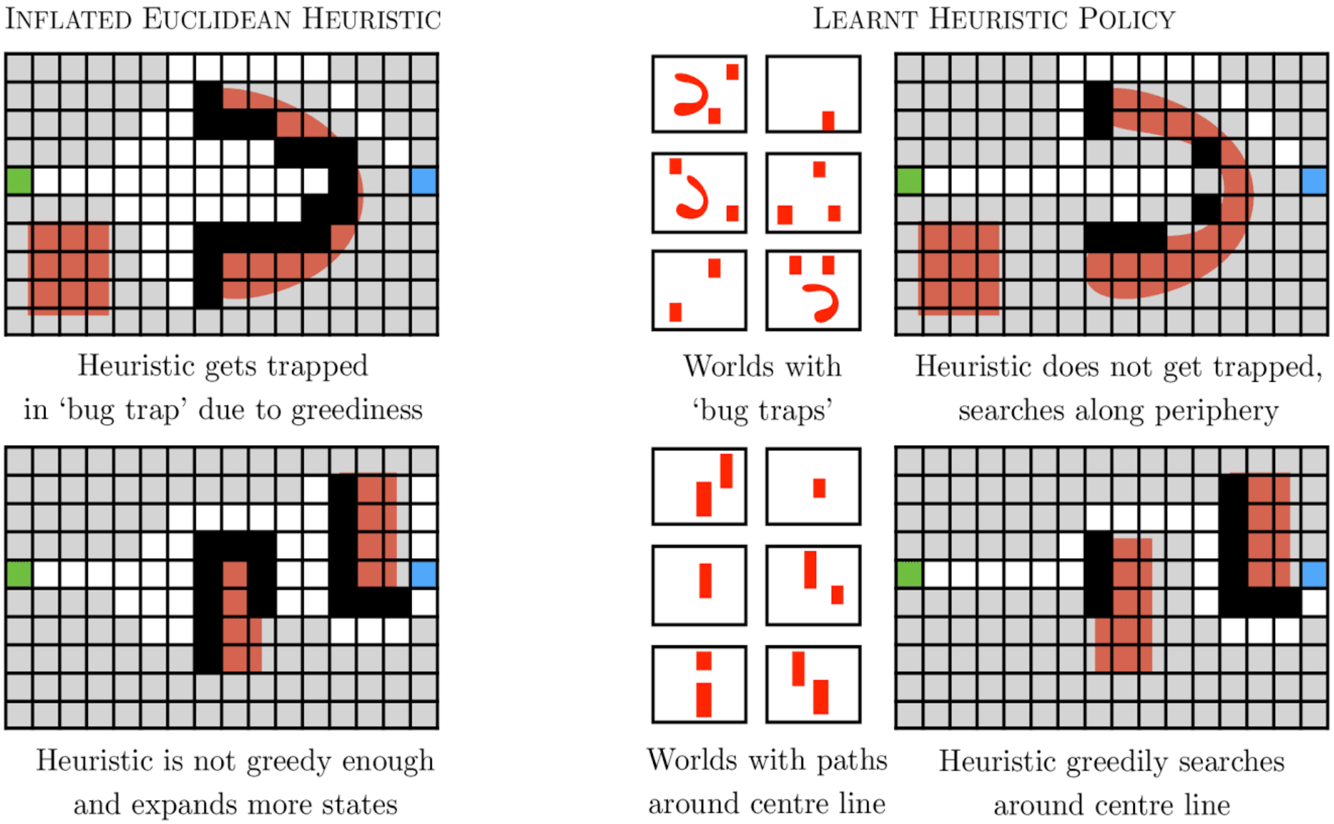
Repeat steps(1-3)
to add km samples

Train $\hat{\pi}_{i+1}$ on $\mathcal{D}$

Repeat steps (1-4)
to train N policies

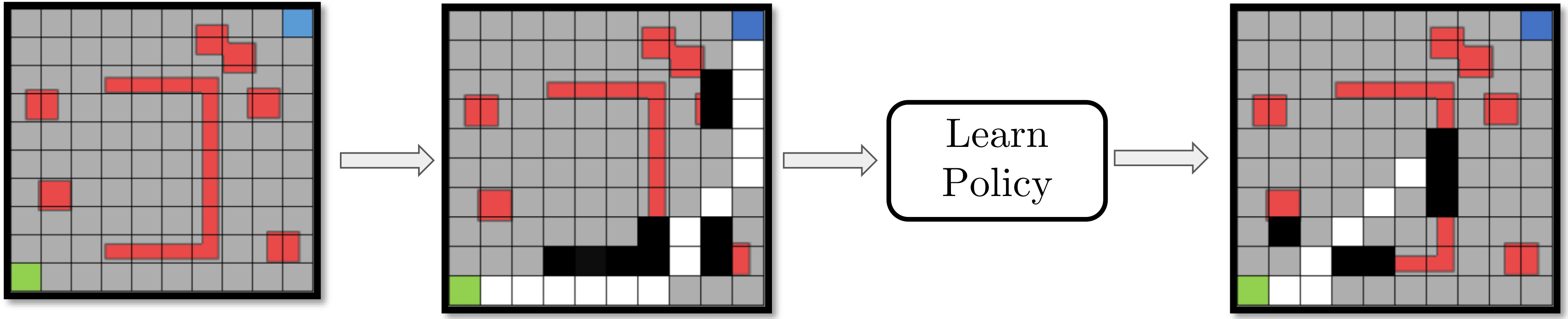
Aggregate data,
update policy
and repeat

Appendix 4: Model-free policy guides planner



Appendix 5: Learning Heuristics via Behavior Cloning

Suffers from distribution mismatch problem



Sampled problem instance(s)

White – Nodes expanded
Black – Invalid neighbors

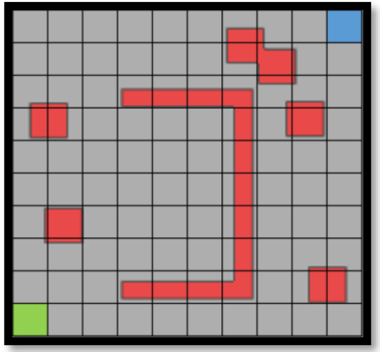
Oracle expands nodes only along least effort path

Data collected on Oracle's state distribution

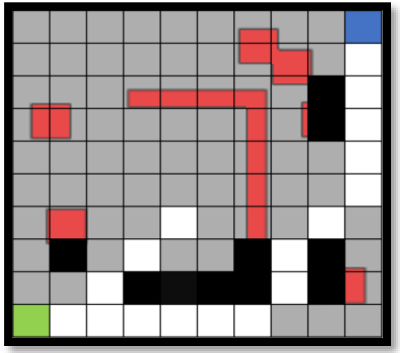
Learner makes mistake and gets lost

Appendix 6: Iterative learning with dataset aggregation

Train on distribution of states encountered by learner (Ross et al., 2011)



Sampled problem instance(s)



Collect data using mixture of learner + oracle

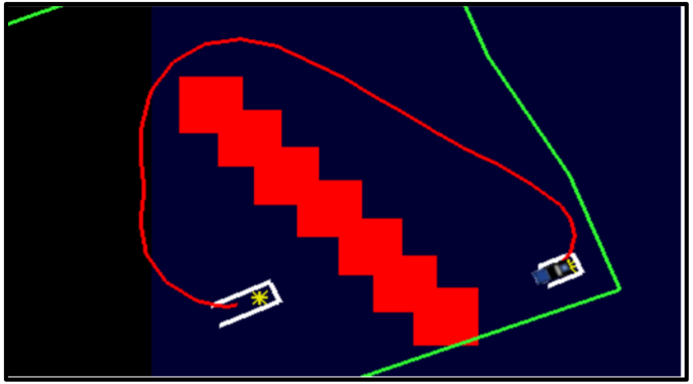
Aggregate data with previously collected data

Update policy using no-regret learning procedure

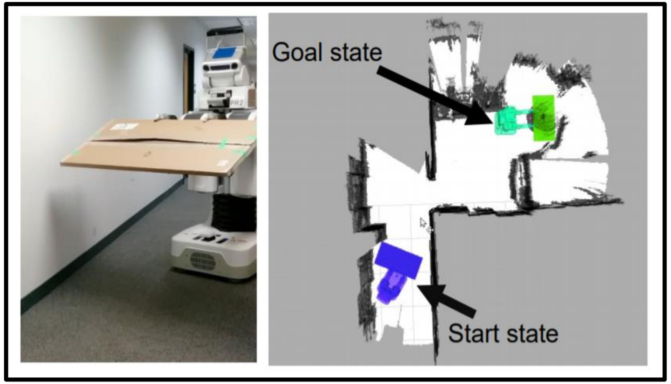
Reduce mixing and iterate

White – Nodes expanded
Black – Invalid neighbors

Appendix 7: Heuristics as distance metrics



Relaxation based approaches
eg. max(Dubin's, 2d Dijkstra)
(Likachev et. al, 2009)



Schedule heuristics efficiently
(MHA*, Aine et al.)

Problems

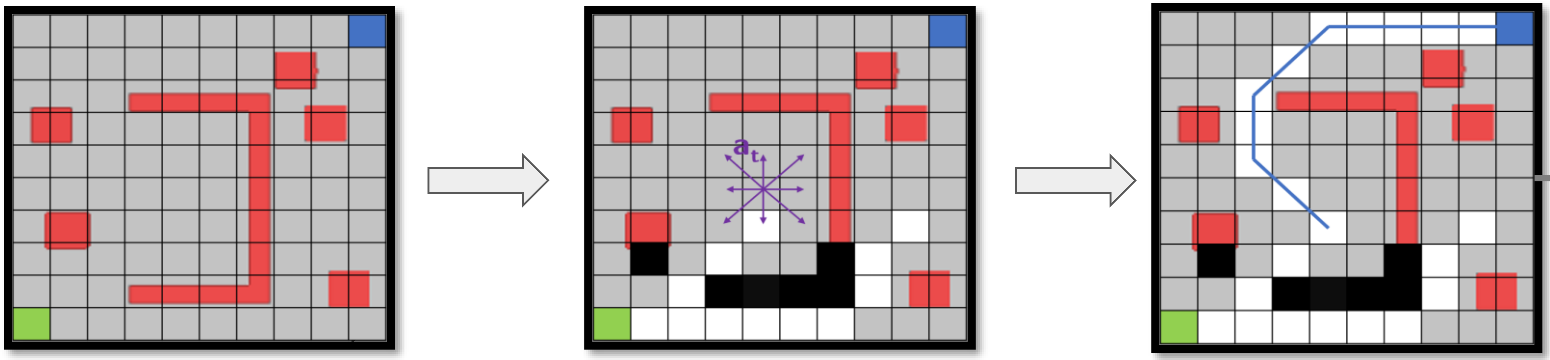
1. Estimating distance metrics can be difficult

2. Minimizing estimation error does not necessarily minimize search effort

Imitation Learning with cost-to-go

When faced with multiple seemingly good actions (as in search), learning policy from optimal demonstrations (0-1 loss) is hard.

Solution: Learn Q-value instead! (AGGREVATE, Ross et al., 2014)



Sampled problem instance(s)

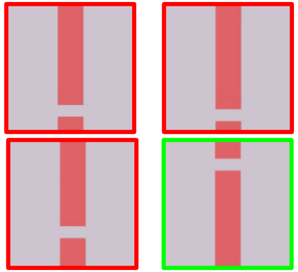
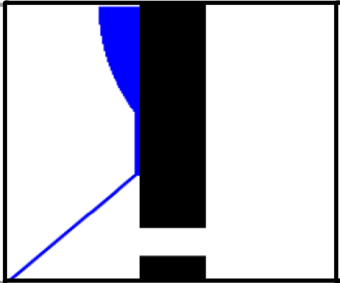
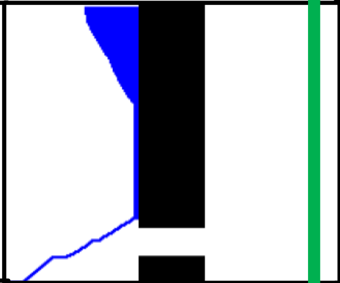
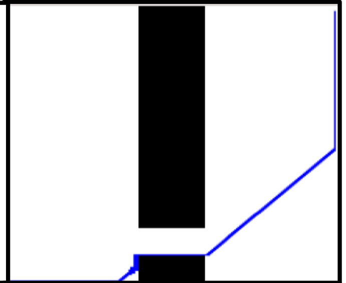
Roll-in mixture policy to uniformly sampled timestep ; choose action a_t

Query/roll-out oracle to get $Q^{OR}(s_t, a_t)$

Aggregate data and update policy as before

Reduce mixing and iterate

SaIL adapts behavior of search in response to change in $P(\phi)$

	Dataset	Greedy (h_{EUC})	Behavior Cloning	SaIL
Alternating gaps: gap 70% at bottom and 30% at top				
Shifting gaps: gap uniformly distributed vertically